GPU-based implementation of Ptycho-ADMM for high performance X-ray imaging

Pablo Enfedaque[1], Huibin Chang[1,2], Hari Krishnan[1], and Stefano Marchesini[1]

[1] Computational Research Division, Lawrence Berkeley National Laboratory
[2] School of Math. Sci., Tianjin Normal University
pablo.enfedaque@gmail.com

Abstract. X-ray imaging allows biologists to retrieve the atomic arrangement of proteins and doctors the capability to view broken bones in full detail. In this context, ptychography has risen as a reference imaging technique. It provides resolutions of one billionth of a meter, macroscopic field of view, or the capability to retrieve chemical or magnetic contrast, among other features. The goal is to reconstruct a 2D visualization of a sample from a collection of diffraction patterns generated from the interaction of a light source with the sample. The data collected is typically two orders of magnitude bigger than the final image reconstructed, so high performance solutions are normally desired. One of the latest advances in ptychography imaging is the development of Ptycho-ADMM, a new ptychography reconstruction algorithm based on the Alternating Direction Method of Multipliers (ADMM). Ptycho-ADMM provides faster convergence speed and better quality reconstructions, all while being more resilient to noise in comparison with state-of-the-art methods. The downside of Ptycho-ADMM is that it requires additional computation and a larger memory footprint compared to simpler solutions. In this paper we tackle the computational requirements of Ptycho-ADMM, and design the first high performance multi-GPU solution of the method. We analyze and exploit the parallelism of Ptycho-ADMM to make use of multiple GPU devices. The proposed implementation achieves reconstruction times comparable to other GPU-accelerated high performance solutions, while providing the enhanced reconstruction quality of the Ptycho-ADMM method.

1 Introduction

Ptychography provides the unprecedented capability of imaging macroscopic specimens at nanometer wavelength resolutions while retrieving chemical, magnetic or atomic information. It was proposed in 1969 with the aim of improving the resolution of x-ray and electron microscopy. Since then, it has been successfully employed in a large array of applications, and shown to be a remarkably robust technique for the characterization of nano materials. For this reason, it is currently used in scientific fields as diverse as condensed matter physics [1], cell biology [2], materials science [3] and electronics [4], among others. Ptychography is based on recording the distribution of the scattering pattern produced by the interaction of an illumination with a sample. In a ptychographic experiment, only the signal intensities are measured, so one has to retrieve the corresponding phases to be able to reconstruct an image of the sample. It falls under the category of phase retrieval problems [5]. In the case of ptychography,

ICCS Camera Ready Version 2018
To cite this paper please use the final published version:
DOI: 10.1007/978-3-319-93698-7_41

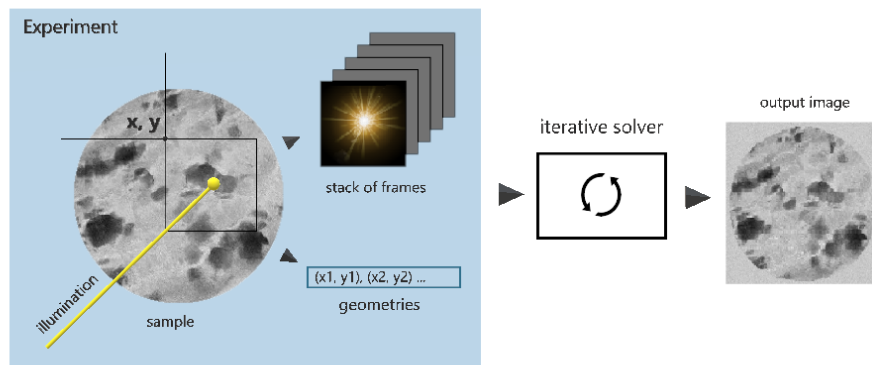

Fig. 1: Overview of a ptychography experiment. An illumination source consecutively scans regions of the sample to produce a stack of phase-less intensities. The stack and the geometry of the measurements are fed to an iterative solver that retrieves the phases and reconstructs an image of the original sample.

the phases can usually be recovered by exploiting the redundancy inherent in obtaining diffraction patterns from overlapping regions of the sample.

From an algorithmic point of view, ptychography reconstruction can be briefly explained as follows (Fig. 1). The input is a stack of multiple frames containing phase-less measured intensities. Each frame corresponds to a snapshot of the light source through a specific region of the sample. These regions are known for each frame, and they are referred to as the geometry of the measurements. Using the stack of frames and their geometries, a non-linear iterative solver repeatedly approximates the phases of the measurements using two constraints: (1) the match between overlapping regions of the frames and (2) the match with a given model for the data. After the solver reaches an exit condition, the output is the overlap of the stack of frames (now with phases) in their corresponding geometries. This overlap corresponds to the 2D reconstructed image of the sample.

Computationally, ptychography poses multiple challenges. The primary challenge is that the stack of measured frames is typically two orders of magnitude bigger than the final reconstructed image. A real case example: a 700×700 pixels image of a cluster of iron particles is recovered from a stack of 900 frames, each one containing 256x256 samples (1:125 output/input ratio). It is also common that the reconstruction algorithms employ additional copies of the measured frames (or additional auxiliary structures of the same size). On the bright side, the algorithms employed in ptychography reconstruction commonly use highly fine-grained parallel operations with few dependencies. This inherent parallelism is usually exploited to achieve reasonable reconstruction times, frequently employing many-core accelerators, such as GPUs [6].

An essential consideration in ptychography algorithms resides in the data models and solver employed. Choosing the proper ones is far from trivial. In a real scenario, models for the illumination source or the background of the measurements are also

ICCS Camera Ready Version 2018
To cite this paper please use the final published version:
DOI: 10.1007/978-3-319-93698-7_41

usually considered. The models and solver employed determine the robustness of the reconstruction (regarding noise or experimental uncertainties), the convergence speed, and the image quality. One of the latest advances in ptychography reconstruction has been recently developed by the CAMERA team at the Lawrence Berkeley National Laboratory (LBNL). The research proposes a new model for data fitting and a new algorithm based on the Alternating Direction Method of Multipliers (ADMM) [7]. The proposed method, referred to from now on as Ptycho-ADMM [8], has been mathematically proven to converge faster than state-of-the-art algorithms, while producing better quality images, and to be more resilient to noise. Ptycho-ADMM benefits come at the expense of increased computational requirements. Besides the input stack, Ptycho-ADMM needs to keep in memory the solution stack and an additional multiplier of the same size, thus handling three times the amount of measured data. The multiplier needs to be updated in each solver step, and it is employed in the optimization of all models, so additional computation is also required.

In this paper we tackle the computational constraints of Pytcho-ADMM and design the first high performance implementation of the method. Ptycho-ADMM parallelism is analyzed to develop a CUDA-based multi-GPU solution that can efficiently make use of multiple GPU devices to achieve state-of-the-art reconstruction times. The performance of the proposed implementation is compared with SHARP [6], a high performance GPU-based ptychography solution. Although the number of arithmetic operations and memory footprint of Ptycho-ADMM is higher than that of solvers employed in SHARP, our implementation is able to achieve comparable reconstruction times, in addition to providing the robustness inherent to the Ptycho-ADMM models. The proposed Pytcho-ADMM implementation is already being used in the microscopes installed in the Advanced Light Source in the LBNL, and the code will be soon available in the Department of Energy online repository system [9].

This paper is structured as follows. Section 2 first overviews the Ptycho-ADMM method and its models, and later reviews the CUDA programming model and the basics of GPU computing. Section 3 presents the proposed solution with a detailed description of the techniques employed, and Section 4 assesses its performance through experimental tests. The last section summarizes this work.

2 Background

2.1 Ptycho-ADMM overview

A ptychography experiment is usually defined as follows. A localized X-ray illumination ω scans through a specimen u, while a detector collects a sequence J of phaseless intensities a. The goal is to obtain a high resolution reconstruction of the specimen u from the sequence of intensity measurements. In a discrete setting, $u \in \mathbb{C}^n$ is a 2D image with $\sqrt{n} \times \sqrt{n}$ pixels, $\omega \in \mathbb{C}^{\bar{m}}$ is a localized 2D illumination with $\sqrt{\bar{m}} \times \sqrt{\bar{m}}$ pixels, and $a_j^2 = |\mathcal{F}(\omega \circ \mathcal{S}_j u)|^2$ is a stack of phase-less measurements $a_j \in \mathbb{R}_+^{\bar{m}}\ \forall 0 \leq j \leq J-1$. The operator $|\cdot|$ represents the element-wise absolute value of a vector, $\circ$ denotes the element-wise multiplication, and $\mathcal{F}$ denotes the normalized 2-dimensional discrete Fourier transform. Each $\mathcal{S}_j \in \mathbb{R}^{\bar{m} \times n}$ is a binary matrix that crops a region j of size $\bar{m}$ from the image u.

ICCS Camera Ready Version 2018
To cite this paper please use the final published version:
DOI: 10.1007/978-3-319-93698-7_41

In practice, as the illumination is almost never completely known, one has to solve a blind ptychographic phase retrieval problem [10], as follows:

$$\text{To find } \omega \in \mathbb{C}^{\bar{m}} \text{ and } u \in \mathbb{C}^n, \quad s.t. \; |\mathcal{A}(\omega, u)|^2 = a^2, \tag{1}$$

where bilinear operators $\mathcal{A} : \mathbb{C}^{\bar{m}} \times \mathbb{C}^n \to \mathbb{C}^m$ and $\mathcal{A}_j : \mathbb{C}^{\bar{m}} \times \mathbb{C}^n \to \mathbb{C}^{\bar{m}}$ $\forall 0 \leq j \leq J-1$, are denoted as follows:

$$\begin{aligned} \mathcal{A}(\omega, u) :=& (\mathcal{A}_0^T(\omega, u), \mathcal{A}_1^T(\omega, u), \cdots, \mathcal{A}_{J-1}^T(\omega, u))^T, \\ \mathcal{A}_j(\omega, u) :=& \mathcal{F}(\omega \circ \mathcal{S}_j u), \end{aligned}$$

and $a := (a_0^T, a_1^T, \cdots, a_{J-1}^T)^T \in \mathbb{R}_+^m$.

Instead of directly solving the quadratic multidimensional systems in (1), Ptycho-ADMM is based on the following nonlinear least squares model:

$$\min_{\omega \in \mathbb{C}^{\bar{m}}, u \in \mathbb{C}^n} \tfrac{1}{2} \big\| |\mathcal{A}(\omega, u)| - a \big\|^2. \tag{2}$$

A mapping $\mathcal{B}(\cdot, \cdot) : \mathbb{R}_+^m \times \mathbb{R}_+^m \to \mathbb{R}_+$ is used to measure the distance between the recovered intensity $g \in \mathbb{R}_+^m$ and the collected intensity $f \in \mathbb{R}_+^m$ as

$$\mathcal{B}(g, f) = \tfrac{1}{2} \|\sqrt{g} - \sqrt{f}\|^2. \tag{3}$$

Based on the the above mapping $\mathcal{B}(\cdot, \cdot)$, a general nonlinear optimization model for blind ptychography similar to (2) can be rewritten as follows:

$$\text{Model:} \qquad \min_{\omega \in \mathbb{C}^{\bar{m}}, u \in \mathbb{C}^n} \mathcal{G}(\mathcal{A}(\omega, u)), \tag{4}$$

with $\mathcal{G}(z) := \mathcal{B}(|z|^2, |a|^2)$. The support or amplitude constraints of the illumination and image [6, 11] can also be incorporated into (4).

To solve (4), Ptycho-ADMM employs an auxiliary variable $z = \mathcal{A}(\omega, u) \in \mathbb{C}^m$, such that an equivalent form of (4) is formulated as below:

$$\min_{\omega, u, z} \mathcal{G}(z), \quad s.t. \quad z - \mathcal{A}(\omega, u) = 0. \tag{5}$$

The corresponding augmented Lagrangian reads:

$$\Upsilon_\beta(\omega, u, z, \Lambda) := \mathcal{G}(z) + \Re(\langle z - \mathcal{A}(\omega, u), \Lambda \rangle) + \tfrac{\beta}{2} \|z - \mathcal{A}(\omega, u)\|^2, \tag{6}$$

with multiplier $\Lambda \in \mathbb{C}^m$, a positive parameter β, $\langle \cdot, \cdot \rangle$ representing the L^2 inner product in complex Euclidean space, and $\Re(\cdot)$ denoting the real part of a complex number. Consequently, instead of minimizing (4) directly, one seeks a saddle point of the following problem:

$$\max_{\Lambda} \min_{\omega, u, z} \Upsilon_\beta(\omega, u, z, \Lambda). \tag{7}$$

Ptycho-ADMM proposes the following update steps to solve the problem in (7), which summarize the method:

ICCS Camera Ready Version 2018
To cite this paper please use the final published version:
DOI: 10.1007/978-3-319-93698-7_41

$$u^{k+1} = \frac{\sum_j \left(\mathcal{S}_j^T((\omega^{k+1})^* \circ \mathcal{F}^* \hat{z}_j^k)\right)(t)}{\sum_j (\mathcal{S}_j^T |\omega^{k+1}|^2)(t)}, \tag{8}$$

$$\omega^{k+1} = \frac{\sum_j (\mathcal{S}_j (u^k)^*)(t) \times (\mathcal{F}^* \hat{z}_j^k)(t)}{\sum_j \left|(\mathcal{S}_j u^k)(t)\right|^2}, \tag{9}$$

$$z^{k+1} = \frac{a(t) + \beta |z(t)|}{1+\beta} \times \operatorname{sign}(z(t)), \tag{10}$$

$$\Lambda^{k+1} = \Lambda^k + \beta(z^{k+1} - \mathcal{A}(\omega^{k+1}, u^{k+1})), \tag{11}$$

given an iteration k and with $\hat{z}^k := z^k + \frac{\Lambda^k}{\beta}$.

2.2 CUDA and GPU computing

GPUs are massive parallel devices composed by multiple SIMD units called streaming multiprocessors (SM). Modern GPUs have up to several dozens of SMs, and each SM can execute multiple 32-wide SIMD instructions simultaneously. The CUDA programming model defines a computation hierarchy formed by threads, warps, and thread blocks. A CUDA thread represents a single lane of a SIMD instruction. Warps are sets of 32 threads that advance their execution in a lockstep synchronous way. Commonly, all threads in a warp are executed simultaneously as a single SIMD operation. Control flow divergence among the threads of the same warp results in the sequential execution of the divergent paths, so it is commonly avoided. Thread blocks group several warps that are executed independently but that can cooperate using synchronization operations to share data. The unit of work sent from the CPU (host) to the GPU (device) is called kernel. The host can launch multiple kernels for parallel execution in one or multiple GPUs, where each kernel is composed of tens to millions of thread blocks.

The GPU memory is organized in three logical spaces: global, shared, and registers. The global memory is typically allocated in the device main memory, and it is visible to all threads in a kernel. The shared memory is only accessible by warps in the same thread block, while the registers are local to each thread. The communication between the threads in a thread block is commonly carried out via the shared memory. The occupancy of the GPU (or of a SM) is the percentage of allocated threads relative to the theoretical maximum. It is constrained by the amount of shared memory and registers assigned per thread. The registers have the highest bandwidth and lowest latency, whereas the shared memory bandwidth is lower than that of the registers. The shared memory provides flexible accesses, while the accesses to the global memory must be coalesced to achieve higher efficiency. A coalesced access occurs when consecutive threads of a warp access consecutive memory positions.

3 Proposed implementation

The main operations involved in the models of Ptycho-ADMM are point-wise parallel, either across the stack of frames, the reconstructed image or a single frame. In this

ICCS Camera Ready Version 2018
To cite this paper please use the final published version:
DOI: 10.1007/978-3-319-93698-7_41

Algorithm 1 Ptycho-ADMM

Parameters: $frames_m[x, y, z],\ coord[z],\ iter_{max},\ tolerance$

1: **allocate** $image[i, j],\ illumination[x, y],$
$frames_s[x, y, z],\ multiplier[x, y, z]$
2: $frames_s = frames_m$
3: $multiplier = 0$
4: **for** k = 0 **to** $iter_{max} - 1$ **do**
5: $frames_s = \boldsymbol{ForwardFT}(frames_s)$
6: $frames_s = \boldsymbol{UpdateFrames}(frames_s, frames_m)$
7: $frames_s = \boldsymbol{InverseFT}(frames_s)$
8: $frames_s = frames_s + multiplier$
9: $illumination = \dfrac{\boldsymbol{SumAll}(frames_s \times \boldsymbol{Split}(image)^*)}{\boldsymbol{SumAll}(\boldsymbol{Split}(|image|^2))}$
10: $image = \dfrac{\boldsymbol{Overlap}(frames_s \times illumination^*)}{\boldsymbol{Overlap}(|illumination|^2, coord)}$
11: $residual = \boldsymbol{ComputeResidual}(frames_s, frames_m)$
12: **if** $residual < tolerance$ **then break**
13: $multiplier = multiplier - (illumination \times \boldsymbol{Split}(image, coord)) + frames_s$
14: $frames_s = (illumination \times \boldsymbol{Split}(image, coord)) - multiplier$
15: **end for**
16: **return** $image$, $illumination$

section we will present and discuss a GPU-based implementation of Ptycho-ADMM that exploits such parallelism.

The overview of the proposed solution is presented in Algorithm 1. The inputs are the measured frames ($frames_m[x, y, z]$) , the coordinates of the measurements ($coord[z]$), the solver maximum iterations ($iter_{max}$) and a given tolerance. The outputs are the final $image[i, j]$ and $illumination[x, y]$ after the solver reaches an exit condition. The $frames_s[x, y, z]$ stores the partial-solution frames, whereas the $multiplier[x, y, z]$ corresponds to the additional variable required in ADMM. The $image$, $illumination$, $frames_s$ and $multiplier$ store complex numbers that represent pairs of intensity and phase values (stored as $float2$). The input $frames_m$ store the original phase-less values ($float$), whereas $coord$ stores pairs of x, y coordinates ($int2$).

The main operations of the proposed solution are highlighted in bold. $\boldsymbol{Split}$ corresponds to the operator $\mathcal{S}_j$, which defines a j subsection of a 2D image, whereas $\boldsymbol{Overlap}$ is the transposed operator $\mathcal{S}_j^T$, which merges all subsections back into an image. $\boldsymbol{SumAll}$ performs an addition across the third dimension of a 3D volume, as follows:

$$forall(x)\ \{ \quad forall(y)\ \{ \quad forall(z)\ \{ \\ output[x, y] = output[x, y] + input[x, y, z] \quad \}\ \}\ \}$$

ICCS Camera Ready Version 2018
To cite this paper please use the final published version:
DOI: 10.1007/978-3-319-93698-7_41

$\boldsymbol{ForwardFT}$ and $\boldsymbol{InverseFT}$ perform z 2D Fast Fourier Transforms (FFT) over a 3D input, where z is the third dimension of the input. $\boldsymbol{UpdateFrames}$ computes the update step in eq. (10), and $\boldsymbol{ComputeResidual}$ calculates the residual between the measured and solution frames. Operators $+$, $-$, * and $|\cdot|^2$ correspond to point-wise addition, subtraction, complex conjugate and complex norm, respectively. The operator $\times$ denotes a point-wise multiplication when both operands are of the same size, or multiple 2D point-wise multiplications when a 2D plane is multiplied with a 3D volume, as follows:

$$forall(x)\ \{\quad forall(y)\ \{\quad forall(z)\ \{$$
$$output[x,y,z] = input1[x,y] \times input2[x,y,z] \quad \}\ \}\ \}$$

The most computational demanding operations correspond to $\boldsymbol{Overlap}$, $\boldsymbol{Split}$ and $\boldsymbol{UpdateFrames}$. In all three functions, the arithmetic intensity[1] is low, so the key performance considerations are the thread-to-data mapping, the device occupancy and the GPU main memory transfers. The ultimate goal is to maximize main memory bandwidth while re-using as much local data as possible. To this end, improving the device occupancy leads to more active threads, while an optimal thread-to-data mapping allows for higher data locality and coalesced accesses, both strategies leading to (potentially) higher main memory bandwidth utilization.

The proposed $\boldsymbol{Split}$ kernel implementation maps all CUDA threads over the output stack of frames. A single thread block is mapped to a frame so that memory is always read and written in a coalesced way. Contrary to $\boldsymbol{Split}$, the $\boldsymbol{Overlap}$ function presents inherent data dependencies: values from different frames can overlap on the same image position. To handle such dependencies, threads are mapped over the input stack and written into the image via atomic additions over main memory. Atomic operations risk serializing multiple high latency operations when concurrency is high, penalizing performance even in latest CUDA architectures. In our scenario, atomic operations provide the best performance compared to more elaborated solutions. This is because the arithmetic load of the $\boldsymbol{Overlap}$ kernel is low, and the latency of the atomic operations can be easily hidden by the main memory transfers.

Data sharing is not required across the solution's main operations. This permits avoiding shared memory to use only register allocation instead, improving in this way the latency of local accesses and the overall occupancy [12]. The thread block size employed is typically 128, which permits optimal theoretical occupancy in current GPU architectures. The mapping of CUDA threads to data employed always guarantees coalesced main memory access, normally using strides of wide equal to the thread block size. To further reduce GPU main memory transfers, some lesser operations are fused into the main CUDA kernels. For instance, basic point-wise arithmetic operations, the illumination multiply or residual computations are usually computed with the nearest $\boldsymbol{Overlap}$ or $\boldsymbol{Split}$ kernel calls. Several kernel fusions implemented in the code are not reflected in Algorithm 1 for illustrative purposes.

Forward and Inverse 2D FFTs represent a significant amount of the pipeline arithmetic computation. FFT GPU implementations have been extensively studied, being

[1] Ratio of number of arithmetic operations computed per memory access.

ICCS Camera Ready Version 2018
To cite this paper please use the final published version:
DOI: 10.1007/978-3-319-93698-7_41

the cufft library one of the most competitive solutions performance-wise. In the proposed implementation, we employ the cufft library to compute $\boldsymbol{ForwardFT}$ and $\boldsymbol{InverseFT}$. To further maximize performance, multiple 2D FFTs are batched together, which permits the library to fusion kernel calls and maximize data re-using.

The above explanation omits multiple minor steps across the whole solving process. Different stabilizers, regularizers, penalization factors, etc. are introduced in some of the models to maximize converge speed and stability. Many of the minor computation steps are implemented using the Thrust library in order to maintain pipeline flexibility and clean interfaces. This necessary tradeoff slightly hinders performance, considering that the ideal case is to fuse all minor computation steps with surrounding kernel calls.

3.1 Multi-GPU solution

The above algorithm and discussion focus on a single GPU implementation. We extend the Algorithm 1 to support multi-GPU execution. The proposed solution employs the NVIDIA Collective Communications Library (NCCL) to implement inter-GPU communication. The partition scheme employed breaks down the workload by means of dividing the different copies of the stack of frames. This way, the $frames_m[x, y, z]$, $frames_s[x, y, z]$ and $multiplier[x, y, z]$ are divided across the z dimension based on the number of GPUs employed.

Almost all operations computed in Algorithm 1 present no dependencies across different frames when processing the 3D stacks. The exceptions are the operations carried out in lines 9, 10 and 11 of Algorithm 1. $\boldsymbol{SumAll}$ performs an addition over the z dimension of a 3D volume, whereas $\boldsymbol{Overlap}$ requires all frames to add their values into the result image. $\boldsymbol{ComputeResidual}$ also have to consider the residuals generated from all independent executions. All three dependencies can be solved in the following way: (1) compute the local partial result, (2) reduce across all partial results (3) broadcast the reduced output to all independent processes. The reduce operation is an addition in all three cases. Step (2) and (3) are implemented using the directive $ncclAllReduce()$, which performs both the reduced addition and the broadcast. Step (1) is implemented in the same way as in the single-GPU execution, but taking sub-sets of frames instead of the whole stacks.

The proposed partition scheme permits a very efficient handling of the data dependencies. Communication is limited to 2D reductions when computing $\boldsymbol{Overlap}$ and $\boldsymbol{SumAll}$, and it is only a scalar reduction when calculating $\boldsymbol{ComputeResidual}$. The amount of communication is in this way comparatively small, with respect to the 3D volumes processed locally. To further reduce communication, we propose an additional optimization: communication can be configured to occur every solver iteration (default) or every n iterations. When $n > 1$, the iterations with no communication employ previous iteration results as non-local data. This can slightly reduce convergence speed, in exchange of increased performance (see next section). During iterations with no communication, the solver can be executed entirely in parallel across all GPUs. The option to enable periodic communication is provided via a command line parameter.

ICCS Camera Ready Version 2018
To cite this paper please use the final published version:
DOI: 10.1007/978-3-319-93698-7_41

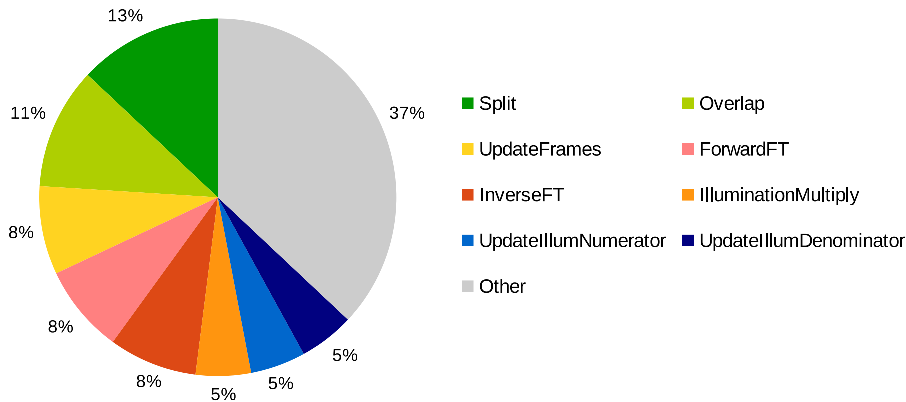

Fig. 2: Percentage of computational time of the main Ptycho-ADMM CUDA kernels when executed on a single GK210B GPU. The input data is a stack of 1600 256×256 frames. Similar results hold for other input sizes.

4 Experimental results

The results presented in this section are executed in a dual socket workstation with two Intel Xeon E5-2683 v4, with a clock frequency of 2.10 GHz and 16 cores each. The workstation is equipped with 4 dual-slot Tesla K80 GPUs, for a total of 8 GK210B devices. Each device has 2496 CUDA cores. The implementations are compiled with gcc 5.4.0 and nvcc 8.0. The profiling results have been obtained with both Nvidia visual and inline profilers, nvvp and nvprof, respectively. All performance results consider the full pipeline execution time, including loading the experimental data, GPU runtime initialization, memory allocation and transfers, and writing back the reconstructed image. The dataset employed corresponds to an experiment performed in the ALS during 2015 that measured a cluster of iron catalyst particles. We have selected different size slices of said experiment to analyze the performance of the proposed implementation with different input sizes. Experimental results presented below hold for other datasets and simulations tested. To simplify the computational analysis, all experiments presented in this section always run 100 solver iterations.

The proposed Ptycho-ADMM implementation achieves a GPU compute utilization of 88%, on average, when executed with significant input sizes (around 100 million input samples). Fig. 2 reports the percentage of computational time of the main Ptycho-ADMM CUDA kernels. *UpdateIllumNumerator* and *UpdateIllumDenominator* compute the numerator and denominator of line 9 Algorithm 1, whereas *IlluminationMultiply* computes the multiplication of an illumination with a stack of frames. *Other* refers to the rest of kernel calls, which have a computational share of less than 5%. A single solver iteration executes a total of 64 CUDA kernels, 42 of which employ less than 0.5% of the total computational time. Out of the kernels with more than 4% of computational

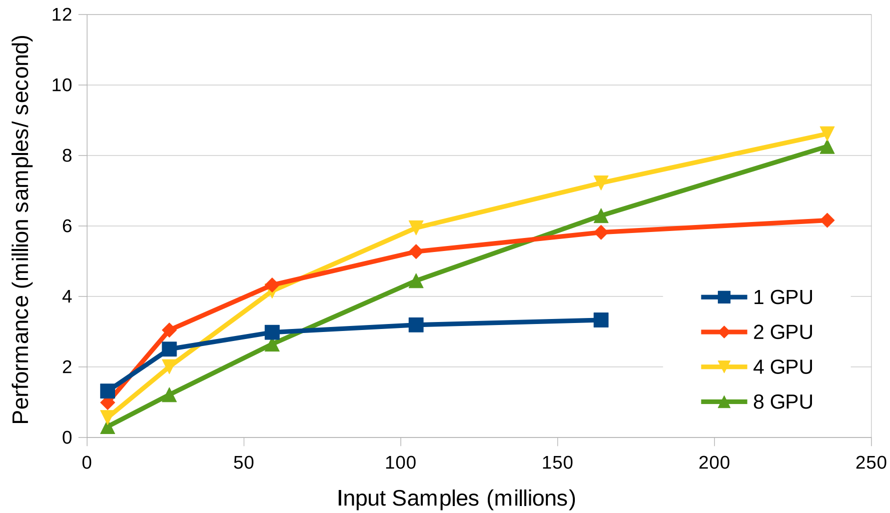

Fig. 3: Performance of the proposed Ptycho-ADMM implementation when executed using 1, 2, 4 and 8 GPUs. Multi-GPU executions communicate every single iteration.

time, the theoretical occupancy is 100%, whereas the achieved experimental occupancy is 96%, on average.

The following experiment assesses the performance and scalability of the proposed Ptycho-ADMM solution for both single- and multi-GPU execution. Fig. 3 shows the performance of the proposed implementation when executed using 1, 2, 4 and 8 GPU nodes[2]. This experiment employs 6 different input sizes. The vertical axis measures performance in millions of input samples divided by total execution time (the higher the better). The horizontal axis corresponds to millions of input samples. The multi-GPU executions presented in Fig. 3 perform communication every iteration.

A horizontal performance line in Fig. 3 represents linear scaling, meaning that the execution time increases proportionally to the input size. Each one of the experiments reported in Fig. 3 presents better-than-linear scaling. This is because the data sizes employed are not big enough to saturate multiple high-end GPU devices, specially with the smaller input sizes. The proposed implementation begins to saturate a single GPU at around 60 millions input samples, although the performance keeps slightly increasing for larger experiments. This proportion holds when executing the solution on 2 GPUs, with a close-to-saturation point at about 200 million input samples. With 4 and 8 GPUs, we can extrapolate the saturation point to be around 400 and 800 million samples. This suggests that, when executed on similar size GPUs, bigger data sets could still benefit from additional multi-GPU performance.

With significant input sizes, multi-GPU executions are 1.7, 2.1 and 1.8 times faster than a single GPU, using 2, 4 and 8 GPUs, respectively. A significant consideration

[2] The experiment with a single GPU and more than 200 million input samples is not reported because it does not fit into the device main memory.

ICCS Camera Ready Version 2018
To cite this paper please use the final published version:
DOI: 10.1007/978-3-319-93698-7_41

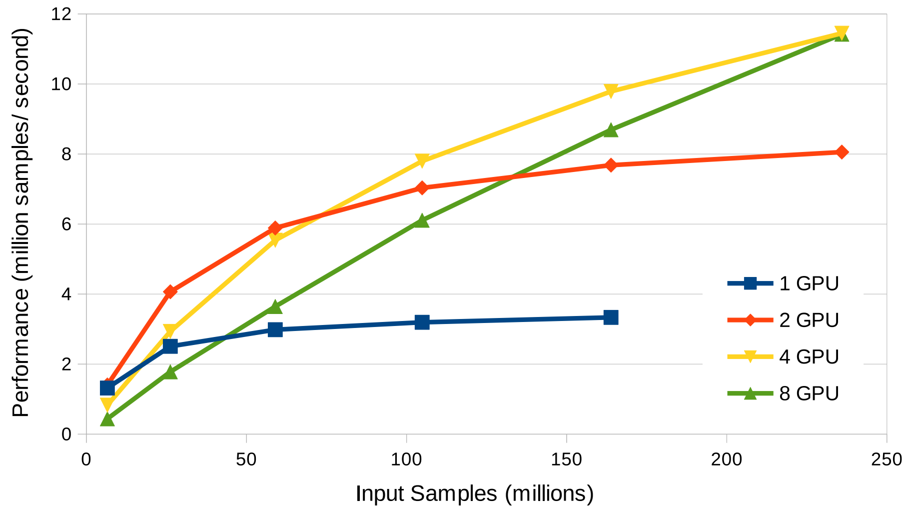

Fig. 4: Performance of the proposed Ptycho-ADMM implementation when executed using 1, 2, 4 and 8 GPUs. Multi-GPU executions communicate every 8 iterations.

in multi-GPU performance resides on the communication frequency employed. The above results can be improved up to a 55% by means of reducing the communication frequency. The tradeoff between communication frequency and solution convergence is maximized when communicating every 8 iterations, on average. When enough iterations are executed, this communication frequency has close-to-no impact on the convergence speed, and significantly accelerates the multi-GPU performance. Fig. 4 presents the same experiment as before, but communicating every 8 iterations. In this experiment the performance of multi-GPU implementations is increased on a 40%, on average, achieving speedups of 2.3, 2.9 and 2.6 respect single GPU, for execution with 2, 4 and 8 GPUs, respectively.

The last test compares the performance of the proposed Ptycho-ADMM implementation with that of SHARP, a GPU-accelerated ptychography solution. SHARP employs the RAAR algorithm [13], a less computational intensive algorithm than Ptycho-ADMM, finely tuned for ptychography reconstruction. The results of the experiment are depicted in Fig. 5, using the same datasets as previous experiments, and executed on a single GPU. The vertical axis represents performance, in millions of input samples divided by execution time (seconds), and the horizontal axis are input samples (in millions). On average, RAAR is 10% faster than the proposed Ptycho-ADMM solution. Besides being extensively optimized for GPU computing, RAAR employs one less additional variable (of the same size of the input stack) and requires one less update step compared to Ptycho-ADMM. On the other hand, the RAAR algorithm does not provide any mathematical convergence guarantee and does not expose the robustness to noise and features proposed by Ptycho-ADMM.

ICCS Camera Ready Version 2018
To cite this paper please use the final published version:
DOI: 10.1007/978-3-319-93698-7_41

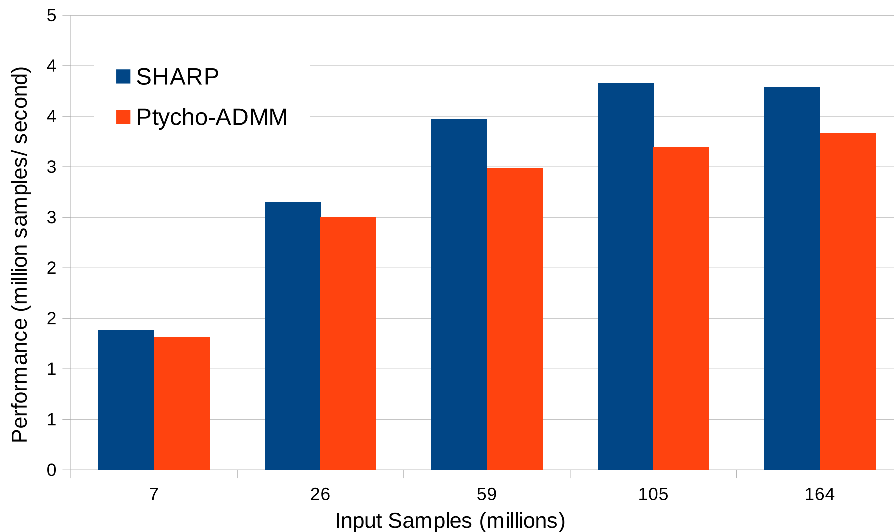

Fig. 5: Performance of the proposed Ptycho-ADMM implementation compared to that of SHARP, both executed on a single GK210B GPU. Different input sizes are employed, ranging from 100 256×256 frames to 2500 256×256 frames. Similar results are obtained with other datasets.

5 Conclusions

This paper presents the first high performance multi-GPU implementation of Ptycho-ADMM. The solution is designed to efficiently exploit the inherent parallelism of the ptychography basic operations. The experimental results show how the implementation is able to saturate multiple high-end GPU devices and to properly scale with the increase of input data size. The ever improving brightness of accelerator based x-ray sources enables novel discoveries by means of providing faster frame rates, larger fields of view and higher resolutions. In this context of continuous increase of input data, scalable reconstruction times and robust solvers that guarantee convergence on a reasonable amount of iterations are highly valuable.

The main future work lines are related to implement a dynamic data feed system that does not require all the data to be allocated (and processed) at the same time. Employing CUDA unified memory could help achieving this goal by means of oversubscribing the GPU main memory. Additional tests with larger datasets (synthetic or real) will also be considered, together with execution on larger scale distributed memory systems using MPI.

Acknowledgment

This work was partially funded by the Center for Applied Mathematics for Energy Research Applications, a joint ASCR- BES funded project within the Office of Science,

US Department of Energy, under contract number DOE-DE-AC03-76SF00098, and by the Advanced Light Source, which is a DOE Office of Science User Facility under contract no. DE-AC02-05CH11231.

References

1. X. Shi, P. Fischer, V. Neu, D. Elefant, J. Lee, D. Shapiro, M. Farmand, T. Tyliszczak, H.-W. Shiu, S. Marchesini *et al.*, "Soft x-ray ptychography studies of nanoscale magnetic and structural correlations in thin smco5 films," *Applied Physics Letters*, vol. 108, no. 9, p. 094103, 2016.
2. K. Giewekemeyer, P. Thibault, S. Kalbfleisch, A. Beerlink, C. M. Kewish, M. Dierolf, F. Pfeiffer, and T. Salditt, "Quantitative biological imaging by ptychographic x-ray diffraction microscopy," *Proceedings of the National Academy of Sciences*, vol. 107, no. 2, pp. 529–534, 2010.
3. D. A. Shapiro, Y.-S. Yu, T. Tyliszczak, J. Cabana, R. Celestre, W. Chao, K. Kaznatcheev, A. D. Kilcoyne, F. Maia, S. Marchesini *et al.*, "Chemical composition mapping with nanometre resolution by soft x-ray microscopy," *Nature Photonics*, vol. 8, no. 10, pp. 765–769, 2014.
4. M. Holler, M. Guizar-Sicairos, E. H. Tsai, R. Dinapoli, E. Müller, O. Bunk, J. Raabe, and G. Aeppli, "High-resolution non-destructive three-dimensional imaging of integrated circuits," *Nature*, vol. 543, no. 7645, pp. 402–406, 2017.
5. S. Marchesini, "Invited article: A unified evaluation of iterative projection algorithms for phase retrieval," *Review of scientific instruments*, vol. 78, no. 1, p. 011301, 2007.
6. S. Marchesini, H. Krishnan, B. J. Daurer, D. A. Shapiro, T. Perciano, J. A. Sethian, and F. R. Maia, "Sharp: a distributed gpu-based ptychographic solver," *Journal of Applied Crystallography*, vol. 49, no. 4, 2016.
7. R. Glowinski and P. L. Tallec, *Augmented Lagrangian and operator-splitting methods in nonlinear mechanics*. Philadelphia, PA: SIAM Studies in Applied Mathematics, Society for Industrial and Applied Mathematics (SIAM), 1989.
8. H. Chang and S. Marchesini, "Blind ptychographic phase retrieval by globally convergent ADMM," *submitted*, 2017.
9. (2018, Jan.) Department of energy online repository system. [Online]. Available: https://github.com/doecode/
10. P. Thibault, M. Dierolf, O. Bunk, A. Menzel, and F. Pfeiffer, "Probe retrieval in ptychographic coherent diffractive imaging," *Ultramicroscopy*, vol. 109, no. 4, pp. 338–343, 2009.
11. R. Hesse, D. R. Luke, S. Sabach, and M. K. Tam, "Proximal heterogeneous block implicit-explicit method and application to blind ptychographic diffraction imaging," *SIAM Journal on Imaging Sciences*, vol. 8, no. 1, pp. 426–457, 2015.
12. P. Enfedaque, F. Auli-Llinas, and J. C. Moure, "Implementation of the DWT in a GPU through a register-based strategy," *IEEE Trans. Parallel Distrib. Syst.*, vol. 26, no. 12, pp. 3394–3406, Dec. 2015.
13. D. R. Luke, "Relaxed averaged alternating reflections for diffraction imaging," *Inverse Probl.*, vol. 21, no. 1, pp. 37–50, 2005.

ICCS Camera Ready Version 2018
To cite this paper please use the final published version:
DOI: 10.1007/978-3-319-93698-7_41